University of West Bohemia

Faculty of Applied Sciences

Department of Cybernetics

bachelor's thesis

KKY/BPIB

Automatic classification of malicious URLs

Author:
Marek Lovčí

Supervisor:
Ing. Jan Švec, Ph.D.

FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Automatická klasifikace škodlivých URL

Zadání

1. Nastudujte současný stav využívání strojového učení pro počítačovou bezpečnost.
2. Zaměřte se na klasifikaci škodlivých URL a popište používané přístupy a dostupné datové sady.
3. S využitím dostupných datových sad implementujte vhodné klasifikátory škodlivých URL.
4. Jednotlivé přístupy otestujte, v práci diskutujte dosažené výsledky.

Automatic classification of malicious URLs

Assignment

1. Learn about the current state of use of machine learning for cybersecurity.
2. Focus on the classification of malicious URLs and describe used approaches and available datasets.
3. Using available data sets implement appropriate classifiers of malicious URLs.
4. Test individual approaches and discuss the results.

Poděkování

Tímto bych rád poděkoval Ing. Janu Švecovi, Ph.D. za odborné vedení, za cenné rady a čas, který strávil čtením a konzultací této práce.

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne ....

Marek Lovčí

Abstrakt

Tato práce shrnuje současný stav využití metod strojového učení v počítačové bezpečnosti. Prezentuje způsoby jak může být strojové učení využito jako obraný mechanizmus proti útokům, včetně příkladů systémů, vytvořených předními společnostmi v oboru počítačové bezpečnosti. Následně je v práci prozkoumáno téma zranitelností a možností útoků na modely strojového učení samotné, spolu s možnostmi zneužití takovýchto technik ve škodlivém *software* za účelem kradení dat, či skrytí přítomnosti *malware* v napadeném systému. V následující kapitole jsou analyzovány techniky klasifikace URL a je implementováno několik klasifikátorů, které rozhodují o škodlivosti, nebo nezávadnosti předložených URL adres. V poslední části je prezentován framework pro vysvětlování závěrů modelů strojového učení.

Klíčová slova strojové učení, počítačová bezpečnost, URL, klasifikace

Abstract

The thesis summarizes current state of machine learning and cybersecurity. The ways how machine learning can be used to protect against adversary attacks are presented, with examples of defense systems constructed by IT security companies. In the same part the topic of machine learning models vulnerabilities and means of exploitation to help malicious software steal data and hide its presence are explored. In the next chapter, the URL classification techniques are analysed and several classifiers are implemented in order to decide on harmfulness or harmlessness of given URLs. In the end, the framework explaining machine learning models' conclusions and its possible applications is investigated.

Keywords machine learning, cybersecurity, URL, classification

Contents

List of Figures

List of Tables

Introduction

Artificial intelligence is one of the most promising technologies of our times [41]. Machine learning methods are adept at analyzing large volumes of data and identifying patterns that might not be apparent to a human. Over time, the efficiency and accuracy of systems implementing AI improves thanks to the ever-increasing amounts of data that are processed and generated by our society. Furthemore, machine learning allows for instantaneous adaptation, without the need for human intervention.

The best examples of machine learning applications can be found in anti-virus software and other types of security applications, which leverage artificial intelligence to implement safeguards to secure users and their data in response to new threats. Thanks to the improvements in computing power and innovation in machine learning methods, the defensive systems can be build with help of image recognition, voice recognition or natural language processing [18]. The gap between the time when a new threat is identified and the system response has been almost eliminated, which is critical for defensive systems.

There are three ways we can look at machine learning and cybersecurity. The first point of view is how we can use machine learning to defend our systems and data. The second way of looking at it is how can be systems, leveraging AI for miscellaneous purposes, contested and what can we do to defend them. The last angle of view if from the attacker's perspective - how can we use machine learning methods to improve attacks and malware.

Weak point of machine learning systems is lack of explainability. This weakness leaves an openings, which can be exploited by hostile methods such as evansion, poisoning and backdoor attacks. Attackers can inject malicious data at the training stage to influence the inference of a model, construct inputs so that model returns inaccurate result and/or extract model parameters or training data from query results [18]. Attacks like evansion, poisoning or backdoor are without a way to get an understandable results very easy to overlook. Not noticing the change in result inference can have fatal consequences resulting in financial loss, credibility loss or even worse.

There comes solution called *model explainer*. Machine learning models are more or less considered to be black-boxes without clear way to justify their decision-making process. It does not represents problem in some application, but especially in the security critical, law or healthcare industries, the model has to be able to provide some form of transparency - in those use cases inexplainability tends to bring legal or business risks. Moreover, enhancing the explainability of artificial intelligence systems helps in analyzing their logic vulnerabilities or dataset blind spots, thereby improving the overall security [18].

In the chapter 1, I will discuss the ML model security risks and convolution of machine learning, cybersecurity and enterpise solutions. Chapter 2 is going to present available datasets and on page 16 will be demonstrated approach for malicious URL detection. The third chapter on page 26 introduces the model explanation framework. As a practical example of use of machine learning for cybersecurity with ability to justify its claims, I am going to make an application in Python programming language using machine learning library *scikit-learn* and LIME (Local Interpretable Model-agnostic Explanations) framework for explaining classification of URL addresses.

Chapter 1

Machine Learning and Cybersecurity

Whether we realize it or not, our lifes have been uploaded. This causes accumulation of big data, which are being used as training sets for Machine Learning methods and technologies such as image recognition, voice recognition and natural language processing. The AI companies are spreading their products to all possible industries including transportation, healthcare or manufacturing. Advancements in ML technologies are made possible by new dedicated hardware - CPU clusters, GPUs and Tensor Processing Units (TPUs), as well as software frameworks and platforms - TensorFlow, Microsoft's Computational Network ToolKit, Keras or scikit-learn.

1.1 AI as a tool for security breaches mitigation

The current generation of security products are incorporating ML technologies. By training cybersecurity solutions on large datasets of network logs and suspicious files, the protection software aims to detect and block abnormal behavior, even if application or dataflow does not exhibit a known malicious signature or pattern [1]. In addition, those solutions can significantly reduce the amount of time needed for detection and response to malicious event. Shortening the discovery times is critical - it takes about 206 days to spot a breach and another 55 days to mitigate caused damage [21]. During that 206-day period, the threat could be performing a range of malicious activities such as stealing sensitive corporate data or confidential information about customers. ML helps security professionals to analyze security alerts in order to identify patterns that may indicate a threat which could be missed otherwise. Without help of AI supported systems, the IT department employees may not use their time efficiently - researching *false positives* and *dead ends*, while missing serious malicious activity.

Nowadays, people spend most of their time on a computer or mobile phone by browsing. If the hacker wants to infect a device, the logical starting point is to create a malicious website or compromise an existing one to carry out the task. The attackers learned how to do URL injections (embeding malicious URLs or entire pages, through the victim page) or malicious redirects to a page usually infected with malware or phishing forms. Furthemore, malware uses for communication and URL leading to a remote control server. All this malicious activity in the system can be prevented by scanning traffic and analyzing network data. Therefore a fast URL classification systems which will automatically block all potential malicious activity of this type is really important. In the chapter 2 we will train such system.

1.1.1 Fast retraining of models

Training of ML model has been, for a long time, the most lengthy and cumbersome part. The frequency and size of the updates to ML models is increassing in order to keep up with offensive attempts - calling out for new techniques to ensure the engine being in accordance with the latest standards and to guarantee security of users. *Avira Operations GmbH* presents parallel training architecture. The first training branch takes around 8 hours to complete. In order to be able to quickly react to new threats, a continuous retraining is performed every 15 minutes [2].

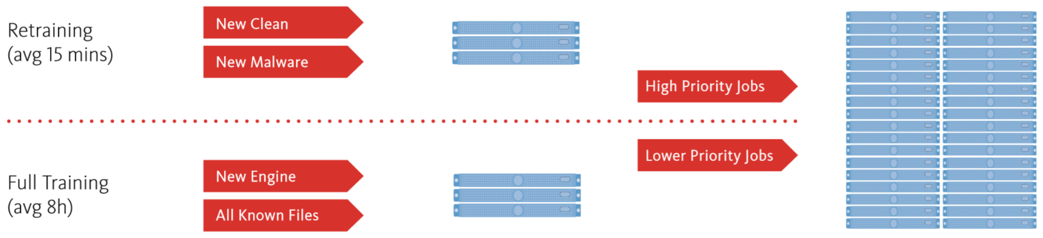
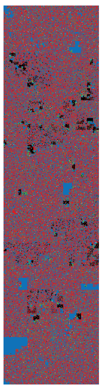


Figure 1.1: Avira's solution to keep security models up-to-date [2]

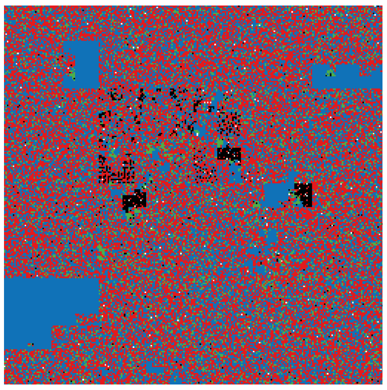
This rapid retraining approach is reducing the vulnerability that potentially exists with systems that are retrained daily or weekly [2].

1.1.2 Ransomware file detection

Different ML approaches can be used for different security objectives. In white paper introduced by CrowdStrike®, company providing cloud-native protection platform for enterprises, they present a robust solution for ransomware detection. By extraction of approximately one hundred features from files and their visualisations (e.g. file size, file entropy, number of sections, distribution of section entropy, imported Dynamic-link library (DLL) names, resource data, embedded IPS/domains, digital signature, etc.) is CrowdStrike®able to train ML model for ransomware file classification [7].



(a) PDF file visualisation



(b) Part of the PDF file visualisation

Figure 1.2: Visualisation of file - feasible for every file. For such data, an analysis similar to image recognition techniques can be performed

1.1.3 Parallel hybrid systems

In the paper "Is machine learning cybersecurity's silver bullet?" ESET's experts sort training data into three groups - malicious, clean and potentially unwanted. They recommend not to use algorithm own output data as inputs, because any further errors are reinforced and multiplied, as the same incorrect result enters a loop and creates more false positives or misses of malicious items [14]. In the next chapter ESET points out how crucial it is to achieve an equilibrium of sufficient protection from malicious items and false positives minimised to a manageable level. Finally, in the chapter *Machine learning by ESET - The road to augur* authors let us take a look under the hood of their ML engine called Augur. For malicious file detection they are using two branches: (i) sandbox analysis followed by advanced memory analysis and behavioural features extraction (these features are later used to train ML models), (ii) ML-based branch. ML-based branch consists of two methodologies: (i) neural networks, specifically deep learning and long short-term memory (LSTM), (ii) consolidated output of six classification algorithms. While consolidating output of those six classification algorithms, two modes (setups) are used. The first one is used for security critical environments, making algorithm more likely to mark file as malicious if most of the previous algorithms vote it as such. The other setup is more conservative - labelling a sample clean if at least one algorithm comes to such conclusion.

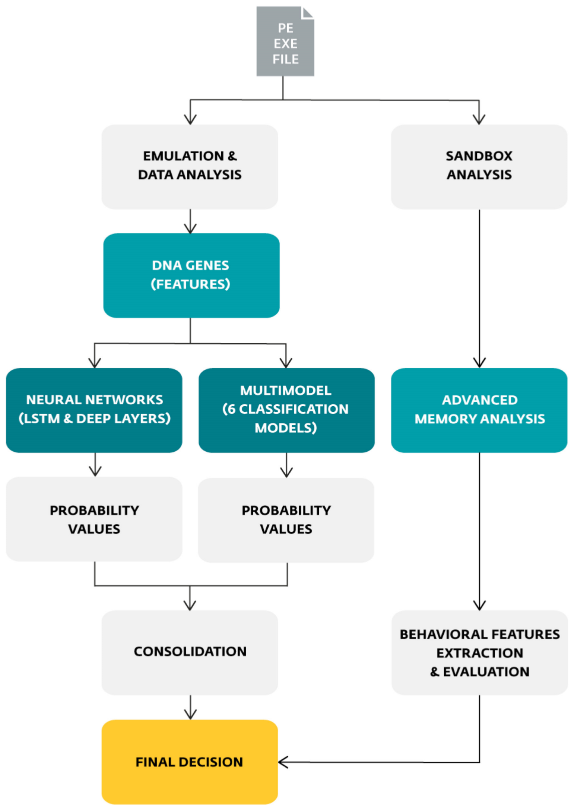

PE EXE FILE
EMULATION & DATA ANALYSIS
SANDBOX ANALYSIS
DNA GENES (FEATURES)
NEURAL NETWORKS (LSTM & DEEP LAYERS)
MULTIMODEL (6 CLASSIFICATION MODELS)
ADVANCED MEMORY ANALYSIS
PROBABILITY VALUES
PROBABILITY VALUES
CONSOLIDATION
BEHAVIORAL FEATURES EXTRACTION & EVALUATION
FINAL DECISION

Figure 1.3: Augur classification model [14]

1.1.4 Serial hybrid systems

Kaspersky Lab identify an average of 360,000 new items of malware every day - approximately 131,400,000 new threats per year [25]. Those security breaches result in direct financial losses, loss of business reputation or financial and legal penalties imposed by regulators. *Kaspersky* provide multi-layer solution consisting of:

(i) exposure prevention (network filtering),

(ii) pre-execution detection based on machine learning,

(iii) runtime control proactively looking out for suspicious behavior of devices in the network (behavioral analysis based on ML),

(iv) automated response such as *automatic rollback* to help restore systems to their pre-attack state, system disinfection techniques or Incident of Compromise (IoC) scanning [25].

Locality-sensitive hashes (LSH) are Kaspersky's solution for malware detection, with ML system used for difficult to detect hashing regions [26]. LSH's goal is to create a reliable fingerprint - a set of features - of a malicious file that could be checked instantaneously [26]. This type of hash function maps very similar files to the same hash values - similar files maps to nearby surroundings. A group of files with a similar hash mapping value is called a hash bucket [26]. Those buckets can be either simple - containing only one type of objects (malware or benign), or hard - containing malware and benign samples. Features are extracted from items in hard regions, to undergo ML classifiaction. Various types of models are pre-trained with human annotated data. Which model is selected for classification of items in region depends on several factors - extractable features, type of objects, etc.

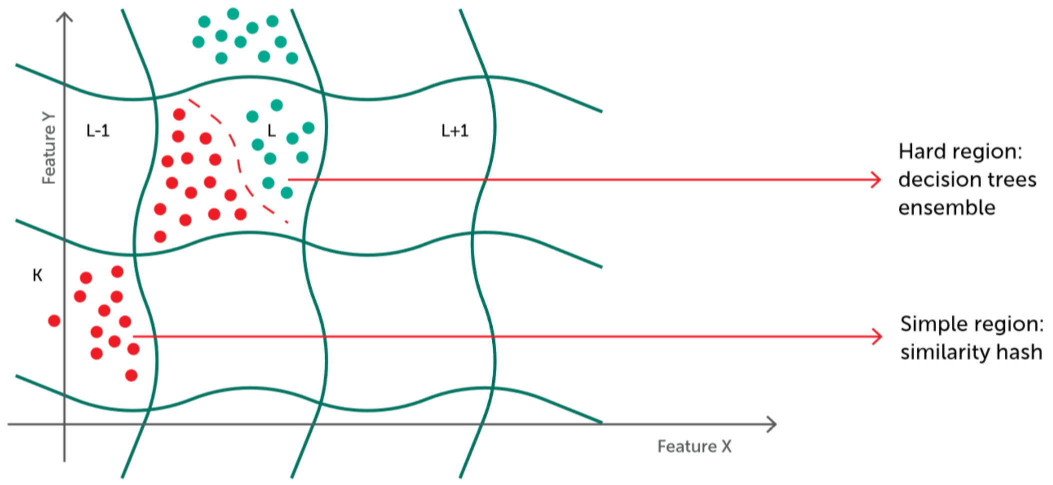

Feature Y
L-1
L
L+1
K
Hard region: decision trees ensemble
Simple region: similarity hash
Feature X

Figure 1.4: Segmentation of object space [26]

1.1.5 Settlement

Circumstances show that although ML-based detection is adequately effective, it needs to be part of a comprehensive solution to address complex security intrusions. At the same time, security companies agree there is no guarantee that classifier can correctly identify all the new samples it encounters, therefore human verification is still needed.

1.2 AI as target of attacks

AI algorithms faces severe security risks which can result in loss of confidence in the government or specific company, not to mention personal safety endanger or property loss. According to the Huawei, there are five security challenges AI system design must overcome [18]. Those five challenges are software and hardware security, data integrity, model confidentiality, model robustness and data privacy.

Firstly, application code and models themselves has to be reliable. There is need for programming languages as well as for processors and GPUs to guarantee memory and thread safety. Since january 2018 vulnerabilities related to modern CPUs fundamental attribute -

speculative execution - became well known by names Spetre (CVE-2017-5753 and CVE-2017-5715), Meltdown (CVE-2017-5754), Foreshadow (CVE-2018-3615) or Spoiler (without CVE number so far). Without solid solutions to mitigate consequences of those issues, attackers may implant backdoors into models to exploit data and construct sophisticated attacks to bring decision models into disrepute. Moreover those attacks are due to limited explainability of AI models difficult to discover.

Second topic relate to data integrity. There is serious concern that attackers can inject data while a model is in the training stage to alter the inference capability or add disturbance into the input samples to change model's interpretation and distort result.

Third challenge involve fear of possible "hijacking" of model through its reconstruction, where data are captured by analyzing large numbers of input and output queries. The fourth one points out insufficient model robustness, originating in training data, which are not good intepretation of real-world data or does not cover all possible edge cases. Those aspects lead to model incompetency to provide trustworthy decisions.

Last risk appertain to data privacy and data leakage through repeated request to obtain users' private information (in scenarios where model is actually trained on the data provided by users). Research from 2013 shows, that using Facebook likes can be used to predict users intimate traits, that are not directly observable in the data. By using ML algorithm you can predict intimate traits like sexual orientation, IQ or drug abuse [23]. This research focused on Facebook likes, although similar research could be done using other kinds of digital footprints, like Spotify playlist, fragments of your Google searches and so on. Possible leakage of data with analogous properties would have serious consequences.

Defending AI systems can become very challenging task. All kinds of methods take an advantage of ML systems vulnerabilities. In order to provide failsafe, data security and high availability guarantees, software architects has to construct system consisting of defense layers for known attacks (which are going to be discussed in the following sections), techniques (e.g. model verification mechanism or explainability of both data and model) for enhancing model robustness and procedures to ensure business security such as model isolation, application redundancy and training data consistency.

1.2.1 Evasion

In security-sensitive applications, samples can be actively manipulated by an profound adversary to confound learning. With the goal to avoid detection, spam emails are often modified by obfuscating common spam words or inserting words associated with legitimate emails [3]. Such activity is called the evasion attack and can be induced either by modifying samples in form of virtual data (e.g. text, audio recording etc.) or by altering real world objects (ammending traffic signs or inducing noise signal). To generate adversarial samples, attackers need to obtain AI model parameters.

In the paper presented by Papernot et al. is proposed technique for generating those examples without knowing exact parameters [35]. They found that data which deceives one model can deceive another model as long as the training data of those two are the same. This so called transferability is used to launch black-box attacks without knowing parameters of the target model. Training data for substitute model are aquired by querying target model several times. After training phase of the substitute, this model can be used to generate adversarial samples, which can be used to deceive the original model. Papernots team archieved admirable results by mischeiving commercial machine learning classification systems from Amazon and Google with approximatelly 96% and 89% success rate respectively, using only 800 queries of the victim model. This demonstration shows that current ML approaches are in general vulnerable to methodical black-box attacks regardless of their structure.

There are several ways to protect models against evasion attack. The first approach is improve model robustness either by **network distillation** or **adversarial training**.

Network distillation is approach used as defense for *deep neural networks.* In principle, one DNN is linked together with the second (executive) DNN in a chain. The classification result from the supplementary DNN is used as training input to the other one. This leads to the transfer of knowledge, reducing sensitivity to adversarial samples and improving the robustness of a model - reducing the success rate of specific attacks such as *Maximal Jacobian-based Saliency Map Attack* [46].

Adversarial training is arguably the simplest method and should be used to train every production-ready model. If model architects have some *a priori* knowledge about possible patterns of adversarial samples, they should use that information to generate some samples and use them to train the model [18]. Training and possible retraining - after obtaining new information about malicious data - leads to more standardized more accurate and robust model [18].

Other option is to append additional component to capture and therefore filter out malicious input sample before it gets into inference stage - **adversarial sample detection**. Simple deterministic detector could be a deterministic comparer having some type of *distance* as a criterion. Detectors vary greatly - forming a group of independent models worth exploring in other papers.

A deformed input samples does not effect normal classification function of a model. **Input reconstruction** works by deforming input samples to defend against evansion attack by adding noise, de-noising, or using an automatic encoder (a type of artificial neural network) [18].

Last but not least method is **model verification**. In general, verification is a discipline of software engineering with goal to assure that software fully satisfies all the expected requirements. Specifically, to verify model means producing a compelling argument that the system will not misbehave under a very broad range of circumstances [15]. To verify ML model a set of legal inputs (similar to test set) has to be defined and verification techniques that guarantee the correctness of the machine learning predictions has to be designed. Simultaneously the verification method has to consider the generalizing capabilities of a model and varying properties of new inputs. Because of those reasons, creating proper verification technique is not an easy task and could be researched as a separate subject.

1.2.2 Poisoning

Data poisoning is a class of attacks on machine learning where an adversary alters a fraction of the training data in order to impair the intended function of the system. Objective can be to degrade the overall accuracy of the trained classifier, escaping security detection or to favor one product over the another. ML systems are usually retrained after deployment to adapt to changes in input distribution, so data poisoning represents serious danger.

Even though many papers focuses on this topic, presents diverse types of attacks and generally agrees that even a small number of malicious training samples are needed to significantly affect the accuracy of models, majority aims their attention at offline learning - not so much on semi-online and online learning. While semi-online mode means that system is learning from data stream but attacker cannot obtain classifiers objective until the end of training, online training means that both the training process and the evaluation of the objective are streamed - example of such system is stock market predicting model. In the article called "Data Poisoning Attacks against Online Learning" researches propose several attack types and feasible defense setups for semi-online and online learning [43]. Performed experiments show that semi-online setting is more vulnerable than the fully-online setting. In the end researchers recommend the use of online methods where classifier does not depends on a just few input samples, so these methods would be less vulnerable to adversarial attacks.

One posibility to protect system against poisoning attack is **training data filtering**. Detecting malicious samples and following purification of training data sets is not a trivial task. In paper from 2016 the researchers presented method called "Curie" for detecting poisonous samples in the dataset [27]. This method works by working in (feature + label) space to filter out malicious samples. Curie is an unsupervised method and an attacker would have to use evasion attacks to pass through Curie. This means that the attacker would have to inject data that works both for evasion attack and poison attack. Such attack would be very complicated to construct.

Regression analysis methods such as linear and ordinary least squares regression are ideal to detect noise and abnormalities in the data sets. Thanks to relative directness presents those methods easy way to fight back data poisoning attack.

Ensemble analysis points out that usage of multiple sub-models - each one of them trained with different training data set - reduces probability of system being affected by poisoning attacks greatly.

1.2.3 Backdoor

Backdoor is in some respects similar to the poisoning attack. Although in this case, attacker is not targeting overall shift in decision inference or accuracy - the model behaves as it should on normal conditions, but on a specific input the output is controlled by adversary, causing malicious behavior when triggered. Corrupting the model so the attack would be feasible can be challenging. Option is to implant some specific neuron into the neural network or to inject carefully crafted samples into the training set. This specific neuron would be called "neural trojan". In general, backdoor is a malicious functionality embedded into the system. The Trojan could have been embedded in the topology, the hardware implementation, or as additional circuitry.

There are againg several ways to mitigate such attack. There is always possibility to try to detect input anomalies or to re-train model (supervised) with legitimate data only, hoping that the Trojans contained in the weights of neural network may be overwritten during the re-training process - requiring the NN to be reconfigurable [31]. **Input pre-processing** aims to prevent the illegitimate inputs from triggering the Neural Trojans by using autoencoder [31].

Model pruning is a popular method for compressing a neural network while keeping normal functions [8, 30]. In addition, this method can be used as a defense mechanism against backdoor attack by possibility of cutting off the "neural Trojan".

1.2.4 Model Extraction

As mentioned in section 1.2.2 ML models are prone to extraction attack - especially those with public API such as *Google Cloud APIs*, *Microsoft Cognitive Service* or *Amazon Machine Learning.* Via analysising input, output and other information about the model, its parameters, training data and structure can be speculated. Those data can be latter used for black-box evansion attack, or just to steal the intellectual propperty the API and model itselves represents. More information about this topic can be found in the paper from 2016, "Stealing Machine Learning Models via Prediction APIs" [37].

Model watermarking embrace the risk of backdoor attack and makes it its advantage. By embeding special neurons, model owners are able to check whether the model was stolen (copied) or not. The disadvantage of this method is obvoius - it is prone to model pruning.

1.2.5 System security

Previous examples of attacks on ML systems and possible defences apply only to specific scenarios. Despite of possibility to combine multiple of these method/technologies in parallel or

Algorithm	Linear	Monotone	Task
Linear regression	Yes	Yes	regression
Logistic regression	No	Yes	classification
Decision trees	No	Some	classification & regression
RuleFit	Yes	No	classification & regression
Naive Bayes	No	Yes	classification
K-nearest neighbors	No	No	classification & regression

Table 1.1: Explainable algorithms [34] - model is **linear** if the association between features and target is modelled linearly; **monotonicity** refers to model's ability to ensure that the relationship between a feature and the target outcome always goes in the same direction over the entire range of the feature, therefore an increase in the feature value leads either to always increase or always decrease in the target outcome

serially, it is not possible to completely defend against all attacks [18]. Therefore it is necessary to prevent possibility of exposure by strengthening model architecture or by supervising and administrating training dataset to mitigate the posibility of data contamination [6]. Such incident would lead to users having a cautious distrust of models reasoning.

Before model deployment, the system should be black-box and white-box tested in order to get some qualitative report about system security [18]. Results from such testing could be refered as **model detectability** - from system theory a system is detectable if all unstable states are observable - in such manner testers check if and how can be system's weaknesses abused.

In the section 1.2.1 the principle of **model verifiability** was already mentioned. Unfortunatelly, for some ML models can be really hard to denote all edge cases. It may be possible to restrict an input range or to apply a saturation function in some cases, but due to complexity and variability of input data, it is not usually viable solution.

It may be difficult to fully interpret models inference process and it is not a functional requirement to be able to do so, for many production systems. On the other hand, there are dozens of solutions, which would benefit greatly from solution which would provide a credible explanation for model inference process. There are handful of solutions introducing **model explainability** to ML and there are overall three categories of possible implementations. Latest literature uses phrase **Explainable AI (XAI)** to capture the essence of the problem.

1. **Explainable data** As Huawei in its paper mentions, if several representative characteristics can be found at data sets and those features are carefully selected, then a some models can be meaningfully interpretted [18]. Of course, data set are not usually simple enough to make such analysis. Moreover, AI model can grow in complexity and even with understandable data and features in the beggining, there is no guarantee that result is interepretable in the end.

2. **Explainable model** Some of the ML models (either for classification or regression) are interprettable naturaly. Their typical properties are *linearity*, *monotonicity* and *interaction features* - possibility to manually add non-linearity into the model. Those models are often way too simplistic for real-world applications, but for an idea those algorithms are sumarized in the table 1.1.

3. **Explainability analysis** The last possibility is to analyze relation between input and output with (or without any) some knowledge about the classifier itselves. This solution is discussed in more detail in section 3.1.

Finally there are security mechanisms based on business requirements [18]. While AI model distribution, the engineers has to ensure architecture and deployment robustness.

1. **Detection:** In some cases it is important to continuously monitor and scan for pottential attack signatures to estimate current risk level [18]. In case of high risk level, the steps are taken to ensure system and data safety.

2. **Failsafe:** Failsafe is a system design feature providing solution to counteract the effect of failure in order to couse the minimal damage to its surroundings. In AI systems, the failsafe mechanism can work by setting up a certainty threshold. When output certainty is lower then threshold, the system can either use recovery procedures, redundant system (e.g. rule-based) or enter the manual processing [18].

3. **Isolation:** Isolation is very basic mechanism to improve security. Functional modules has to be separeted and their connection channels should provide mechanism to verify information validity and reliability.

4. **Redundancy:** In a security and infrastructure critical applications is indispensable to ensure system high-availability and robust decision inference process. A multi-model architecture can reduce the possibility of the system being compromised by a single attack, improving the robustness of the entire system [18].

1.2.6 Data sets security

In addition to model and architecture security mechanism, we have to consider security of data sets. Data often contains personal information of users - so called *sensitive attributes.* Such information can be in a form of personal identifiers or quasi-identifiers. *Personal identifier* is an unique information that identifies a user - a birth number, bank account number and other types of personal IDs. *Quasi-identifiers* are characteristics which needs to be used in combination with others to identify an entity - examples are gender, postal code, age or nationality.

To prevent data stealing and following re-identification of users, several models exists to protect personal information of individuals in dataset. Those privacy models are **optimal k-anonymity**, **l-diversity**, **t-closeness** and **differential privacy**. Three general types of attack to datasets exists: (i) re-identifying an individual, (ii) query whether an individual is a member of a dataset, (iii) linking an individual to a sensitive attribute.

Optimal k-anonymity protects against both cases (i) and (ii) by transforming quasi-identifiers so that at least $k-1$ members of set are indistinguishable from each other - group based anonymization. Identifiers are transformed by suppression (needless attributes are replaced with *dummy* values) and generalization (individual values of attributes are replaced with a broader category - e.g. specific age can be replaced by a range). As k increases risk of data exploit reduces, on the other hand data quality decreases - we are talking about a *privacy-utility* tradeoff. Moreover, the k is limit - in order for this method to work if k is set to $k \triangleq 10$ then any group must contain at least 10 individuals. The first drawback of k-anonymity is vulnerability to *homogeneity attack* which works on premise of data having sensitive value identical within a set of k records - it is enough to find the group of records, the individual belongs to, if all of them have the same sensitive value. Second drawback is the possibility of *background knowledge attack* where attacker identifies associations among one or more quasi-identifiers and reduces the set of possible values for the sensitive attribute.

Both **l-diversity** and **t-closeness** are group based anonymization techniques building on a concept of **optimal k-anonymity**. In addition to **optimal k-anonymity**, **T-closeness** transforms quasi-identifiers such that each group is within a distance t of the distribution of sensitive values for the entire dataset [11]. The distance is measured as the cumulative absolute difference of the distributions, as t decreases both risk of sensitive attribute disclosure and data quality decreases. Suppose that the sensitive attribute is salary. Each group's frequency distribution of salary will be within a distance t from the salary frequency distribution for the entire dataset [11].

Differential privacy and its variants (epsilon, epsilon-delta) are statistical techniques aiming to protect data against *differentiated attack*. The model guarantees that even if someone has complete information about 99 of 100 people in a data set, they still cannot deduce the sensitive information about the final person [47]. The mechanism works by adding random noise to the aggregate data, leaving only a trend without possibility to figure out exact values in data (e.g. information that $n\%$ of users prefer some product over another).

Some other techniques to keep dataset secure exist. In some cases with very sensitive data, the data can be exported from database in a form of already vectorized samples. There is very little to be mined from data in such format. On the other hand if data are not secured by any technology whatsoever, the risk rises. A model may inadvertently or implicitly by design store some of its training data. In this case careful analysis of the model may reveal sensitive information. Solutions like **Private Aggregation of Teacher Ensembles** - PATE in shortcut - exist to adress this problem [18]. *PATE* works by segmenting training data into multiple sets, each for training an independent ML model [18]. Those models are used to cooperatively train a student model by voting. It is assumed that an adversary cannot access teacher models - he can access only student model. The student learns to predict an output chosen by voting among all of the teachers [18]. The student is trained with public (not sensitive) data. Model's privacy properties are indisputable - adversaries are not able to extract sensitive data with access to the student model only.

1.3 The malicious use of ML in cybersecurity

As machine learning provides protection against attackers, attackers themselves can exploit machine learning for malicious use. The most obvious malicious use of ML is information gathering, which can be used for social engineering and personalized attacks (such as more sophisticated and believable phishing emails). Another type of usage can be to break through Google's reCAPTCHA which is test build specially to tell human and a machine (a bot) apart [20]. ML methods can be certainly used with attacks mentioned in the section 1.2 - poisoning, model extraction or evasion attack as presented in paper by *Weiwei Hu* and *Ying Tan*, "Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN" where authors describe how they built a *generative adversarial network (GAN)* based algorithm, that will be able to bypass ML detection protection system [17].

Nowadays, a sophisticated ML supported malware imitating human attacker's behavior starts to appear. The expert human adversary try to blend into their target environment as much as possible, in order to remain hidden. To achieve "invisibility", the attacker understands what normal behavior of the infected system looks like and adjusts his techniques accordingly [10]. It is possible for malware to acquire this contextual understanding and integrate into a target environment autonomously [10].

All following techniques are just usages of user and process behavior analysis, but represents real threats and trends in malicious software development. In the research white paper "The Next Paradigm Shift: AI-Driven Cyber-Attacks", cybersecurity company *Darktrace* presents three inovative attack scenarios, in which attackers used ML for more successfull infiltration. Each threat covers different phase of the attack cycle - lateral movement, C&C traffic and data exfiltration [10].

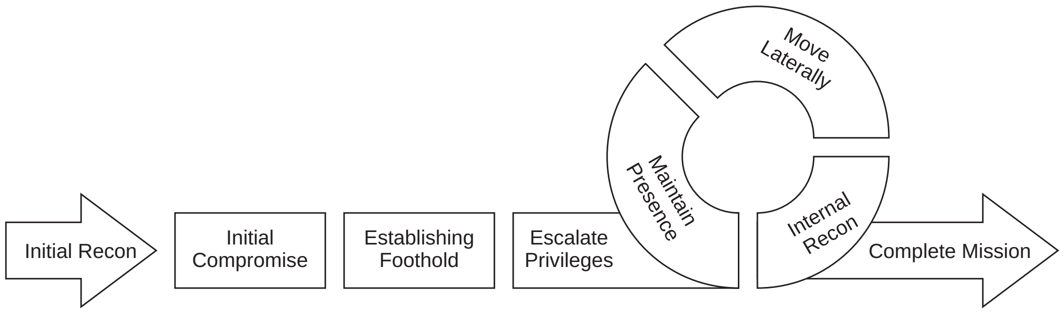


Figure 1.5: Malware attack lifecycle [9]

1.3.1 Trickbot: smart assault method selection

First case represents a modular malware *Trickbot* - a banking *Trojan* type malware targeting Windows machines. *Trickbot* malware is being developed and improved for several years now (at least from 2016), and was extended by numerous additional capabilities, such as:

- achieving persistence (through scheduled tasks),
- disabling Microsoft's Windows Defender (built-in antivirus),
- gathering email addresses and sending out spam,
- collecting system and memory information, user accounts, lists of installed programs and services,
- fingerprinting browsers and capturing data from them (including passwords),
- steal passwords from Microsoft Outlook and FTP applications like WinSCP or Filezilla,
- spreading itself to other computers on the same network by exploiting SMB vulnerabilities with the EternalRomance exploit [42].

Besides SMB protocol exploits, *Trickbot* is spread via email attachments, malicious URLs, Microsoft Word documents with macros and other malware like Emotet. The stolen information is exfiltrated to C&C servers and used to achieve continuous access to infected networks and to deliver ransomware - e.g. Ryuk, specialized in targeting enterprises. Version of *Trickbot* discovered by *Darktrace* installed program called **Empire**, the PowerShell post-exploitation agent. This allows the attackers to conduct manual intrusions, for high-value targets [10].

At least two potential and dangerous ML-based improvements of Trickbot exist. Malware could autonomously utilize multiple payloads for monetization – for example stealing banking details and locking machines for ransom [10]. This functionality is already available, but its modularity encourage to use ML and intelligently choose which payload will yield the highest profit based on the context of the environment and infected machine [10]. Second improvement is to analyse target environment and select best self-propagation process. Ability to choose lateral movement techniques accordingly makes communication with C&C server unnecessary (until need to send information), making attack stealthier, more difficult to detect and therefore more dangerous [10].

1.3.2 Intelligent evasion

Next case shows malware ability to analyze its environment to remain hidden for longer periods of time. At a utility company, the malware used a variety of stealth tactics and obfuscation to stay hidden [10]. Malware was downloaded to the device from the Amazon S3 cloud platform - attacks exploiting Software-as-a-Service (SaaS) like Google Apps Script to deliver malware via URLs are quite common.

The malware showed ability of blending into the environment - utilizing its own self-signed SSL certificate for windowsupdate.microsoft.com and made HTTP(S) requests to an attacker-controlled IP address utilizing the fake Windows certificate [10]. In order to remain unseen, the communication took place over the *well-known ports* like 443 for Hypertext Transfer Protocol over TLS/SSL (HTTPS) and 80 for Hypertext Transfer Protocol (HTTP), therefore blending in with regular network traffic.

According to Open Source Intelligence (OSINT) it is expected, that most malware will be improved with ML-based capability to blend into its environment, without need to be highly targeted. This skill will come in forms of establishing C2 channels during regular business hours, communication over the well-known ports, using domain names for C2 communication and data exfiltration that closely resemble the target's own domain or company name or domain fronting for popular content delivery networks [10].

Malware will learn about its environment via user and process bahavior analysis to gain an understanding of what communication will be most effective in the target's network.

1.3.3 Slow data exfiltration

The third and the last scenario presents malware with very obscure behaviour. In this case, the malicious software was stealing information from attacked system in an exceptionally slow pace to avoid triggering the data volume threshold in security tools [10]. The malware was sending packages less than 1MB in size each, for the extended time period [10]. It is difficult for security system to identify such a small data flow as malicious, especially in a bandwidth extensive infrastructure, even though the malware does use abnormall IP addresses and ports [10].

As mentioned in the previous case, ML powered malware constitute a serious threat while low-and-slow data exfiltration is difficult for traditional tools to detect. Knowledge of the context of target's environment will cause even more problems, while malware could change data volume threshold dynamically, based on the total bandwidth used by the infected machine [10].

Chapter 2

Malicious URL detection and classification

Whether you use the internet for business purpose or personal use, you can be a victim of a malware attack. In order to prevent infection, security companies use tools like Domain Name System filtering to ensure, the users will not become victims of malware. With large percentage of malicious URL links found on "good" domains and techniques like Domain Generation Algorithms, which does not require a significant amount of sophistication while still being highly effective, it is impossible to capture all (or at least signifficant amount) vicious URLs without an automated system. While there exists no simple rule, which would clearly distinguish between malicious and benign link, the machine learning system is necessary.

Detection of malicious URLs is effective when performed in real time, detects new URLs with high accuracy, and recognizes specific malicious activity type (e.g. phishing, spam, adware, drive-by-downloads etc.). Nowadays, security vendors rely on filtering unencrypted DNS traffic based on signature detection (DNS firewall and blacklisting) to examine DNS queries and filter out known malicious domains. This approach has its flaws. Blocking is based on *hostname*, which does not allow propper blocking of defaced URLs which contains both malicious and legitimate web pages. Furthemore, in several years most of the DNS traffic will be encrypted, preventing any spying, spoofing or man-in-the-middle attacks, making the traditional entry-analyzing methods incapable to block malicious content [12]. There comes the ML solutions that can learn themselves on various types of datasets to identify malicious activity or anomalies within the web traffic. Those datasets can be either bi-class (e.g. "bad" or "good" URLs) as presented in this paper or multi-class (e.g. malware, spyware, ransomware or fakenews, gambling sites and porn sites - depending on the requirements). Other notable use of ML based URL classification is by proxy servers, which can provide filtration of complete URLs. DNS and Proxy blocking can be performed both server and client side.

2.1 Data

The simplest way how to acquire data for URL classification would be to use some type of data which the system already collects. Network monitoring is a common practice in all sorts of environments - therefore use of network monitor dump files, for example *pcap* files would come in handy. It is possible to obtain the full URLs from HTTP payloads together with other information like complete HTTP headers, the problem arises if the communication is encrypted by the SSL protocol. The only information we can acquire from *HTTPS captures* is hostname because almost every other usefull information is encrypted. With HTTPS the path and query string of the URL is encrypted, while the hostname is visible inside the SSL *handshake* as plain text.

The information like *how long is the hostname registered* or *website content* would be possible to get. The methods used to acquire such information are very time and data consuming, while security system has to be fast and modest in data usage, therefore downloading web contents of every analyzed URL or calling additional query is not possible.

It would be best to use only the data we have available without additional information retrieval. Furthemore the URLs are often constructed to deceive users (more information in the section 2.1.1), therefore we want our features to be those exact characteristics.

2.1.1 Origins of malicious URLs

While doing malicious URL classification, I have to mention, where do malicious URLs originate. Some of the URLs are used for scam or phishing. Such addresses are usually very similar to the original ones, e.g. *goggle.com* instead of *google.com* or *arifrance.com* instead of *airfrance.com* - this practise is called **typosquatting** [5].

$$\underbrace{www}_{\text{subdomain}}\ .\underbrace{\overbrace{verylongdomainname}^{\text{domain name}}.\ \overbrace{academy}^{\text{top-level domain}}}_{\text{root domain}}$$

Another similar practise known as **domain squatting** (also known as **cybersquatting**) consists in registering *top-level domain* with *second-level domain* belonging to some company. Such domains are either sold to the company which owns the trademark or can be used with mischievous intentions.

> For example, the name of your company is “abcompany” and you register as abcompany.com. Then phishers can register abcompany.net, abcompany.org, abcompany.biz and they can use it for fraudulent purpose [5].

Another approach is to generate addresses using Domain Generation Algorithm (DGA). DGAs are used to produce a large quantities of domains that are going to be used for communication to the malware's command and control servers [40]. It is common practise for an DGA to generate 1000 domains which creates noise in the network. Main objective of a hacker is to register one of those addresses, which will become an active Command & Control (C&C) server. Infected system then uses this address to communicate and receive commands from the attacker [29].

Address created by DGA usually looks similar to this one: *gwhhpnrfkdiedhga.ki* - pseudo-random set of characters, but other algorithms are more ingenious - combining established techniques with word dictionaries (e.g. *101paypal-login.in*).

DGA domain detection can be complicated. Some techniques use Shannon entropy or n-grams, but those are most effective in recognising pseudo-random addresses. To recognise the other ones, we need different approach - machine learning algorithm combining above-mentioned techniques [22, 29, 24].

2.1.2 Datasets

For classifier training which will distinguish between malicious and harmless (benign) URLs, the dataset with URL and class (with labels *good* and *bad*) will be needed. Probably the largest available dataset is available from University of California San Diego at the http://www.sysnet.ucsd.edu/projects/url/. The data was made available as part of a research project, containing data from 120 days and each observation has approximately 3,200,000 features. The target variable contains 1 if it is a malicious website and -1 otherwise. Large dataset like this one would be ideal. Unfortunatelly, the data are already in a form of matrices of features, therefore feature extraction is not possible, making dataset unsuitable for this study.

To be able to produce vectorizer and feature matrix, raw data are necessary - URL in its raw string form, with label denoting class affiliation. Datasets meeting those conditions can be found quite easily, although some of them are rather specific, labeling malicious URLs cording to the type of exploitation - malware, spyware, ransomware, etc. Those sets can be used, but even more suitable ones can be found.

In the practical example is used a combined set of URLs from sources mentioned in the table 2.1 and table 2.2. **Hosts** is a repository which consolidates reputable *hosts* files, creating list of potentinally unwanted websites focusing on fakenews, gambling, porn and general malicious URLs. The Kaggle.com is website offering ML challenges and datasets for general public, all supported by the Kaggle community.

For the classifiers' training a **balanced** dataset was generated, containing total of 125,000 URLs - 50% malign, 50% benign, with minimal URL length of 5 characters, average URL length of 45 characters, maximum URL length of 2307 characters, mode of 31 and median of 35 characters.

Dataset	Download date	URL
Kaggle	28th Semptember 2018	https://www.kaggle.com/antonyj453/urldataset#data.csv
Hosts	12th March 2019	http://sbc.io/hosts/alternates/fakenews-gambling-porn/hosts

Table 2.1: Information about datasets (name, download date and download URL)

	Number of samples			Length of URL				
Dataset	Total	Malicious	Benign	Minimal	Maximal	Average	Mode	Median
Kaggle	411,247	18%	82%	6	2307	48	31	41
Hosts	55,575	100%	0%	5	93	18	17	18

Table 2.2: Information about datasets (name, number of samples and URL statistics)

The final training dataset is a *csv* file in the format as in the table 2.3, with the header *url, label.* For the testing purposes, four other dataset were made. Those sets are made in order to simulate real-world network traffic, where data are strongly biased. The first set is unbiased, the second one consists of ten times as much benign as malicious samples. The third set has hundred times and the fourth has thousand times more benign samples. In the concrete numbers:

1. 300 malign samples to 300 benign samples;
2. 300 malign samples to 3000 benign samples;
3. 300 malign samples to 30,000 benign samples (refered to as "set with 1 : 100 ratio");
4. 300 malign samples to 300,000 benign samples.

URL	Label
https://www.kino24.kz/blog/engine/modules/plugin/source.php?id=	bad
https://www.warcraft-lich-king.ru/wp-admin/present.php?origin=lobs&amp;session=NzEwMTU2MDkwNDM0Nzk	bad
https://phxfw.wordpress.com/2011/11/19/phxfws-own-to-judge-miss-arizona-teen-pageants/	good
http://www.vertor.com/torrents/117148/Macy-Gray-The-Very-Best-Of-Macy-Gray	good
https://www.mycomicshop.com/search?TID=174061	good

Table 2.3: Dataset example

The development data were created from the dataset with 1 : 100 ratio by spliting the set into 80% *test* and 20% *dev* set.

2.2 Text feature extraction

Text feature extraction is a process of tokenizing, counting and normalizing string data to be fed to the machine learning algorithms. In general, only a few groups of features for URL classification are commonly used.

- **URL-Based Features** are extracted from URL. Examples are *total length of address*, *number of digits in URL*, *legitimate name of brand in subdomain* or *number of subdomains.*
- **Domain-Based Features** consist of information about domain registrant and domain itself, e.g. *how many days passed since the domain was registered* or if *is the registrant name hidden.*
- **Page-Based Features** are harder to obtain, then the others. Such features are obtainable from tools like **Google Analytics** - *average visit duration*, *number of pages visited*, etc. In most cases, PageRank is the easiest metric to get.
- **Content-Based Features** are collectable with technique called **web scraping**. Data are extracted from website and analysis is performed.

The paper focuses on the use of *URL-based* features (specifically *word occurences*) - while the dataset does contain already blocked and inactive URLs, there is no possibility to extract *domain-based*, *page-based* or *content-based* features. At the same time, the use of *URL-based* features is potentially safer (there is no need to download anything from the website) and represents an interesting challenge to classify a URL with minimal memory and performance requirements.

The whitepaper “Heuristic-based Approach for Phishing Site Detection Using URL Features” published in 2015 describes the Korean team’s solution for phishing site detection using URL features [28]. The team presents 26 features which are used in the detection mechanism they created.

First group of features is related to Google search engine suggestions. First three features are based on similarity of primary domain, subdomain and a path to Google suggestions. The other three features checks whether searched term (domain, subdomain and path) is presented in a whitelist. Levenshtein distance between the two terms is calculated. If a search term is similar to the suggested one, then it is more likely to be marked as a suspicious, because that site may be used as a trap when user misspells a word/address [28]. If the Levenshtein distance equals to zero (both terms are the same), then domain is added into the trustworthy whitelist, such site is probably legitimate.

The are three features in the second group, which are extracted through page ranking. PageRank is an algorithm which offers a way of measuring the importance of a website. Malicious websites usually have a very low page rank value because such sites are not visited by many people and they exists only for a short time [28].

Third group is about analysing the structure of URL address and its patterns. URL rarely contains some special characters and legitimate sites usually have one “top-level domain”. Therefore many TLDs can signify a fraudulent site.

In the next group URL property values are checked. Temporary malicious URLs often does not use HTTPS protocol and does not have DNS or WHOIS record.

Lastly length of a subdomain is calculated and phishing terms in the URL are checked.

In the end the research team compared several algorithms on the sample of 6000 URLs. In the conclusion they evaluate “random forest” classifier as the best, for their case. Unfortunately model parameters are not mentioned, and their training and testing data are not available, therefore we are not able to reproduce their outcome.

2.2.1 Tokenization

In our case, string tokenization of URL is done with *wordninja* module, which works by modeling the distribution of the output, assuming all words are independently distributed [16]. In order to properly tokenize continuous chain of words (also known as *string* without spaces), we need to have a list of words sorted from high to low by frequency. The relative frequency of words in a given language can be calculated for example from Wikipedia articles dataset. For each word in the frequency set, the "word-cost" is calculated with the equation (2.1), where N is the total number of words and $k \in \{1, 2, \ldots, N\}$ is the rank of word in dataset.

$$wordcost = \log\left(k \cdot \log(N)\right) \tag{2.1}$$

Let s to be an URL. The dynamic programming is used to infer the position of the spaces. As the first step, the *cost array* is build, containing one number for each character in the original string s. This array is created by iterating over characters in s, calculating the minimum cost of the first i characters, based on *word-cost*, value of charater at the position $i - 1$ and every possible contiguous subsequence of characters with common upper border, given by the index i.

After that, the *cost array* is backtracked to recover the minimal-cost string. Array of characters, represented by s is iterated once again, but backwards. The minimum cost of the set of characters ending with $j := length(s)$ is retrieved (that is why the *cost array* was needed), together with number of characters l, which make up the minimum value. The interval of characters from index $j - l$ till j is one of the words, we were looking for. Now a new ending character is set to $j = j - l$ and *backtracking* continues until j is zero.

The advantage of this solution is that the implementation consumes a linear amount of time and memory [16]. The disadvantage, on the other hand, is the necessity to have the corpus of words sorted by relative word frequency similar to what will the tokenizer actually encounter (e.g. the correct language), otherwise the results will be very bad [16]. Implementation of the algorithm is available at the appendix B.

In the end, the four special tokens are removed - TLD *com*, subdomain *www* and protocols *http*, *https*. They are present to such an extent that it would be very inappropriate to draw conclusions based on their presence.

$$\text{brownfoxjumpsoverdog} \rightarrow (\text{brown}, \text{fox}, \text{jumps}, \text{over}, \text{dog})$$
$$\text{0ZStYTgvFu1U85XxLeE9} \rightarrow (0, \text{z}, \text{sty}, \text{tgv}, \text{fu}, 1, \text{u}, 85, \text{xx}, \text{lee}, 9)$$

2.2.2 Vectorization

For counting and normalizing I am going to use *scikit-learn's* TF-IDF vectorizer. TF-IDF stands for *term-frequency times inverse document-frequency*, it is a statistical measure to evaluate the importance of a word to document in a corpus (2.3). In the equation, $\text{TF}(t, d)$ is how many times that term t occurs in a document d - this is called *raw count*. By default normalized term-frequency is used by *scikit-learn* (2.2). There are other options that can be used like boolean frequency where $\text{TF}(t, d) = 1$ if t occurs in given d and 0 otherwise, or logarithmically scaled frequency where $\text{TF}(t, d)$ is replaced with $1 + \log(\text{TF}(t, d))$.

$$\text{TF}(t, d) = \frac{\text{number of times } t \text{ appears in } d}{\text{total number of terms in } d} \tag{2.2}$$

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t) \tag{2.3}$$

$$\text{IDF}(t) = \log \frac{1 + n}{1 + \text{DF}(t)} + 1 \tag{2.4}$$

Inverse document-frequency is computed as in the equation (2.4) where n is is the total number of documents in the corpus. The function DF (t) is the number of documents in the corpus that contain term t. The TF-IDF vectors are then normalized by the Euclidean (L2) norm (2.5).

$$v_{norm} = \frac{v}{||v||_2} = \frac{v}{\sqrt{{v_1}^2 + {v_2}^2 + \cdots + {v_n}^2}} \tag{2.5}$$

As can be seen, *TfidfVectorizer* has two usefull properties: (i) it lowers the weight of common words, (ii) applies Euclidean normalization after computing the TF-IDF representation.

Implementation of scikit learn TF-IDF vectorizer consists of two parts. As mentioned in the section 2.2.1, each URL in corpus tis splitted into individual words - tokens. The **CountVectorizer** then converts a this collection of words to a matrix of token counts (2.6).

$$\begin{array}{c}\text{This is the first sentence.}\\ \text{This sentence is the second sentence.}\\ \text{And this is the third one.}\\ \text{Is this the first sentence?}\end{array} \Rightarrow \begin{array}{c}\text{and}\\ \text{sentence}\\ \text{first}\\ \text{is}\\ \text{one}\\ \text{second}\\ \text{the}\\ \text{third}\\ \text{this}\end{array} \Rightarrow \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1\\ 0 & 2 & 0 & 1 & 0 & 1 & 1 & 0 & 1\\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1\\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{2.6}$$

An encoded matrix is returned with a length of the entire vocabulary and an integer count for the number of times each word appeared in the collection.

In our case (word occurences in URLs), this matrix is extremely sparse (most of the elements in matrix is 0). The total count of features in the presented dataset is around 1,150,000, while each URL has only a few tokens (in the order of units).

The other constituent part of TF-IDF vectorizer is TF-IDF transformer. This transformer converts a count matrix to a normalized *term-frequency* or *term-frequency times inverse document-frequency* portrayal. We are going to prefer tf-idf instead of the raw frequencies of occurrence of a token, because the objective is to scale down the importance of words that appear more frequently in a given corpus. Informative function of such tokens is (verifiable by experience) lower than descriptive value of words that occure less often in the training set.

Eucledian rescaling means that the each representation of URL has L2 norm equal to 1, therefore length of URL (the number of tokens) does not change the vectorized representation [32].

The vector of token frequencies for a given URL is called a sample [36]. Process of counting and normalizing is called vectorization, the result is **Bag of Words** representation. **Bag of Words** is a strategy of describing a document (in our case an URL) by word occurences ignoring the relative position of the words [36].

2.3 Classifiers

There are many options which classifier to choose for text classification. To name a few supervised learning options [44]:

- **Support Vector Machines**,
- **Logistic Regression**,
- **Naive Bayes**,
- Neural Networks,
- Hidden Markov models,
- Decision Trees,
- Random Forests,
- Boosting and Bagging algorithms,
- k-Nearest Neighbour algorithm.

Each one of them has advantages and disadvantages, which determine how and when can be those methods used. Point of this work is not to train algorithm with high classification success rate. The chosen classifiers were chosen in order to present a wide scale of possible solutions with relatively high classification rate without need to “tweak” model hyperparameters.

The problem, the paper faces, is text binary classification with large sparse matrix as an input. With respect to those criteria, I chose three suitable classifiers - SVM with stochastic gradient descent method, logistic regression and multinomial Naive Bayes.

During the research, I tried others, like decision trees and random forests, but those two generally do not work well with sparse data. There are some exceptions and although DT and RF can work well with sparse data if specific parameters (e.g. number of trees and number of features evaluated at each split decision) are set in a reasonable way, they were labeled as “not suitable” for this task.

KNN algorithm is used to classify instance by finding the k nearest matches in training data and then using the label of closest matches to predict the category. Traditionally, distance such as euclidean is used to find the closest match [13]. The method can use several algorithm to search for points in k-dimesional space like *k-d tree* or *ball tree*, unfortunatelly the brute-force search is used by *scikit-learn’s* implementation of algorithm, which is memory inefficient and raises MemoryError on my computer for the task, not being able to complete training.

2.3.1 Stochastic gradient descent

Stochastic Gradient Descent is a popular optimization technique, which can be used with many learning algorithms. *SGDClassifier* is a linear classifier (linear SVM) that uses Stochastic Gradient Descent for training. In addition it requires less memory and allows incremental (online) learning [36]. Support vector machines are effective in high dimensional spaces and use only a subset of training points in the decision function (called support vectors), so it is memory efficient [36].

Gradient Descent is an iterative method, used to find the values of the parameters of a function that minimizes the cost function as much as possible. The difference between Gradient Descent optimization and SGD is that the second one uses only a few *randomly* selected samples instead of the whole data set for each iteration. Using the whole dataset is useful for getting to the minima in a less noisy or less random manner, but when datasets are humongous (very large), it becomes computationally very expensive [38].

Disadvantage is that SGD requires a number of hyperparameters and is sensitive to feature scaling [36]. The *scikit-learn* implementation requires `loss="log"` parameter to be used in order to enable the `predict_proba` method, which gives a vector of probability estimates $P(y|x)$ per sample x. Method `predict_proba` is required by LIME framework (introduced in chapter 3). The SGD classifier supports multi-class classification by combining multiple binary classifiers into one.

The model is given a set of training examples $(x_1, y_1), \ldots, (x_n, y_n)$ where $x_i \in \mathbf{R}^m$ and $y_i \in \{-1, 1\}$ [36]. The goal is to figure out a linear scoring function $f(x) = w^T x + b$ with parameters $w \in \mathbf{R}^m$ and intercept $b \in \mathbf{R}$ [36]. The prediction is made by looking at the sign of $f(x)$. Model parameters are found by minimizing the regularized training error (2.7), where L is a loss function that measures model fit, R is a regularization term that penalizes model complexity and $\alpha > 0$ is a non-negative hyperparameter [36].

$$E(w, b) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \alpha R(w) \tag{2.7}$$

Model in the following example uses *logistic regression* as a loss function L and the regularization term is the default one (L2 norm) $R(w) := \frac{1}{2} \sum_{i=1}^{n} w_i^2$.

2.3.2 Logistic regression

Logistic regression is a linear, naturally interprettable model for classification and is an extension of the linear regression model for classification problems. *Scikit-learn's* implementation can fit binary, One-vs-Rest, or multinomial logistic regression [36].

Input values $\mathbf{x}$ are linearly combined using weights β to predict an output value y.

$$y = \frac{exp(\beta_0 + \beta_1 \cdot x_1 + \ldots + \beta_k \cdot x_k)}{1 + exp(\beta_0 + \beta_1 \cdot x_1 + \ldots + \beta_k \cdot x_k)} \tag{2.8}$$

The β_0 is the bias, $\beta_1, \ldots, \beta_k$ where $k \in \{1, 2, \ldots\}$ are the coefficient that must be estimated from the training data. This is done using *maximum-likelihood estimation* - a minimization algorithm which obtains the parameter estimates by finding the parameter values that maximize the likelihood function (2.9) [4].

$$\mathrm{lf}(\mathbf{x}) = \frac{1}{1 + exp(-\mathbf{x})} \tag{2.9}$$

Logistic regression requires for data to meet several requirements [4]:

- binary output variable,
- noiseless data (since the method assumes no error in the output variable),
- absence of correlated inputs (the model can overfit if multiple highly-correlated inputs are present).

If all assumptions of the linear regression model are met by the data, there is a guarantee to find optimal weights [4].

The disadvantages are oversimplifying resulting in "not that good" predictive performance and risk of *complete separation*, occuring in the case in which there is a feature that would perfectly separate the two classes (weight for that feature would not converge), although there is no need for machine learning in that scenario.

2.3.3 Multinomial Naive Bayes

Naive Bayes is a set of methods based on applying Bayes' theorem.

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots x_n \mid y)}{P(x_1, \ldots, x_n)} \tag{2.10}$$

To reduce the number of parameters, we make the "naive" Bayes *conditional independence assumption.* We assume that attribute values are independent of each other given the class [33].

$$P(x_i|y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|y) \tag{2.11}$$

Multinomial distribution is a generalization of the binomial distribution. Multinomial Naive Bayes is algorithm implemented for multinomially distributed data, suitable for classification with discrete features (word counts for text classification).

Although Bayes method is decent classifier, it is a bad probability estimator, therefore the probability outputs from `predict_proba` are not accurate. Naive Bayes algorithms are mostly used in sentiment analysis, spam filtering or recommendation systems. For document classification problem (whether a document belongs to the category of music, technology, sport,...), Multinomial Naive Bayes is mostly used.

The data distribution is parametrized by vectors $\theta_y = (\theta_{y1}, \ldots, \theta_{yn})$ for every class y where n is the size of the vocabulary and θ_{yi} is the probability $P(x_i \mid y)$ of feature i appearing in a sample belonging to class y [36]. The parameters θ_y are estimated by the equation (2.12).

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n} \tag{2.12}$$

The number of how many times the feature i appears in a sample of class y in the training set T is denoted by $N_{yi} = \sum_{x \in T} x_i$ [36]. The $N_y = \sum_{i=1}^{n} N_{yi}$ is the total count of all features for class y [36]. The smoothing coeficient $\alpha \geq 0$ accounts for features not present in the learning samples and prevents zero probabilities in further computations [36].

Other types are Bernoulli and Gaussian Naive Bayes. The Bernoulli Naive Bayes classifier assumes that all our features are binary, therefore not suitable for text classification with tf-idf features. Gaussian Naive Bayes (2.13) is used in cases when all features are continuous (assumption of the normal distribution) - not made for cases where features can be represent in terms of their occurrences (text classification) [45].

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right) \tag{2.13}$$

2.4 Evaluation

The main metrics used for evaluation are *precision*, *recall* and *f1-score*; where TP = True positive, FP = False positive, FN = False negative and TN = True negative.

$$precision = \frac{TP}{TP + FP} \tag{2.14}$$

$$recall = \frac{TP}{TP + FN} \tag{2.15}$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2.16}$$

The *precision* is the ability of the classifier not to label as positive a sample that is negative. The *recall* is the ability of the classifier to find all the positive samples. The F1-score can be interpreted as a weighted average of the precision and recall, where the score reaches its best value at 1 and worst score at 0.

In the real world, malicious traffic is always hidden in much larger set of daily, benign data flow. The question, how the trained classifiers perform on those heavily biased data sets arises. Dataset bias is a problem beyond the scope of this paper, yet, let me examine consequences of bias for URL classification.

Models were trained using *scikit-learn's* default parameters, with the currated balanced dataset. The accuracy and precision, recall and F1-score for malicious classification are in the table 2.4, *support* is the number of samples from the testing dataset that lie in that particular class. The data for the report in the table 2.4 is the testing part (the 80%) from the set with 1 : 100 ratio, the same data were used to create ROC curve in the figure 2.1. The ROC curve is a performance measurement for classification at various thresholds settings, giving the information how much is model able to distinguish between classes.

Classifier	Precision	Recall	F1-score	Support
SGD	0.11	0.92	0.19	239
LRC	0.15	0.97	0.26	239
MNB	0.11	0.97	0.19	239

Table 2.4: Combined classification report for Stochastic gradient descent (SGD), Logistic regression (LRC) and Multinomial Naive Bayes (MNB) classifier with the default threshold = 0.5 for class *bad*

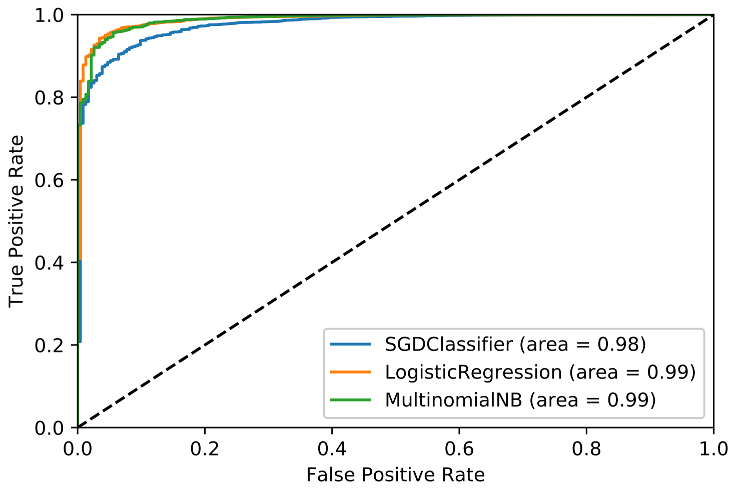


Figure 2.1: Receiver operating characteristic (ROC) curve for given classifiers

The *scikit-learn* use a default threshold of 0.5 for binary classifications. From the development part (the 20%) from the set with 1 : 100 ratio, the figures 2.2, 2.3 and 2.4 were created, in order to view precision, recall and F1-score at various thresholds. Therefore the ideal threshold for particular data and classifier combination can be determined.

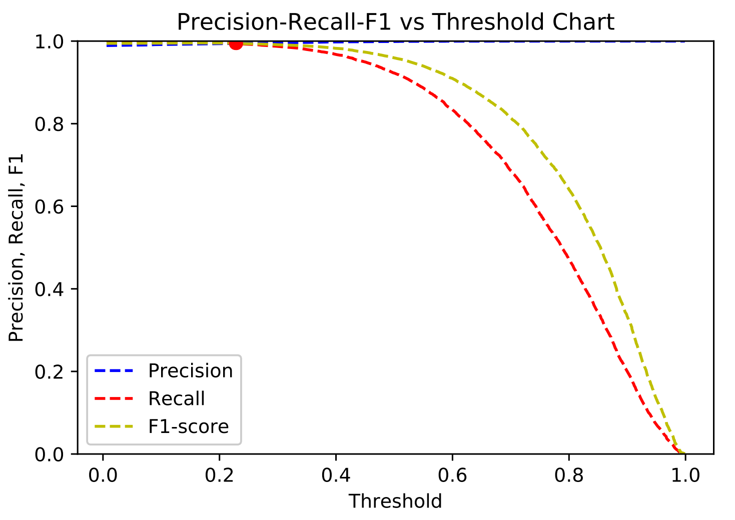


Figure 2.2: Precision, recall and F1-score at various thresholds for SGD classifier

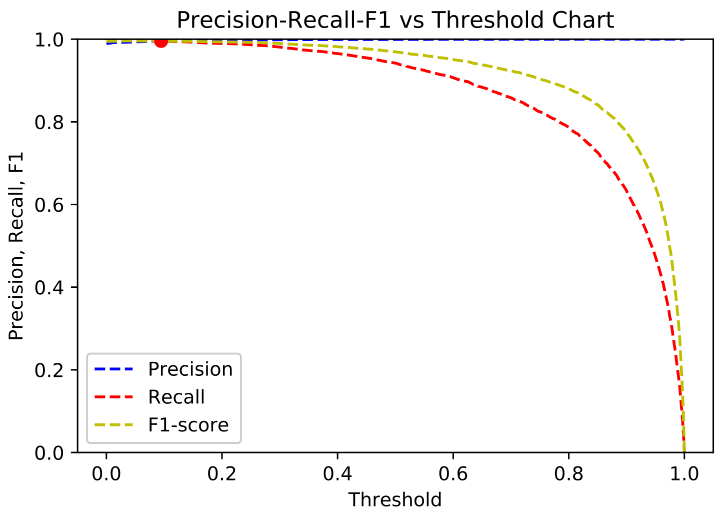


Figure 2.3: Precision, recall and F1-score at various thresholds for Logistic regression classifier

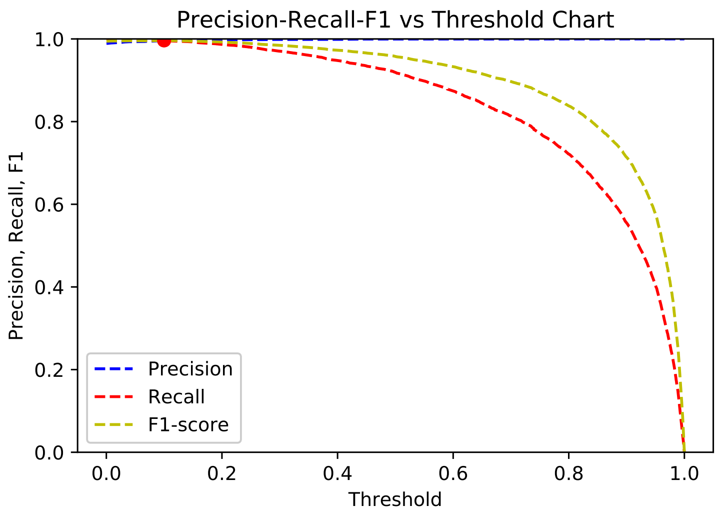


Figure 2.4: Precision, recall and F1-score at various thresholds for Multinomial NB classifier

Now we know that the default threshold (0.5) is not optimal, therefore the new classification report with optimal thresholds is generated.

Classifier	Threshold	Precision	Recall	F1-score	Support
SGD	0.217	0.55	0.53	0.54	239
LRC	0.098	0.65	0.64	0.64	239
MNB	0.110	0.67	0.68	0.67	239

Table 2.5: Combined classification report for Stochastic gradient descent (SGD), Logistic regression (LRC) and Multinomial Naive Bayes (MNB) classifier with the optimal thresholds for malicious class

For the last evaluation test case the four testing sets, each one with different distribution ratio, were used. The results are in the figure 2.5. The results shows the higher the ratio, the lower the optimal threshold and lower the F1-score.

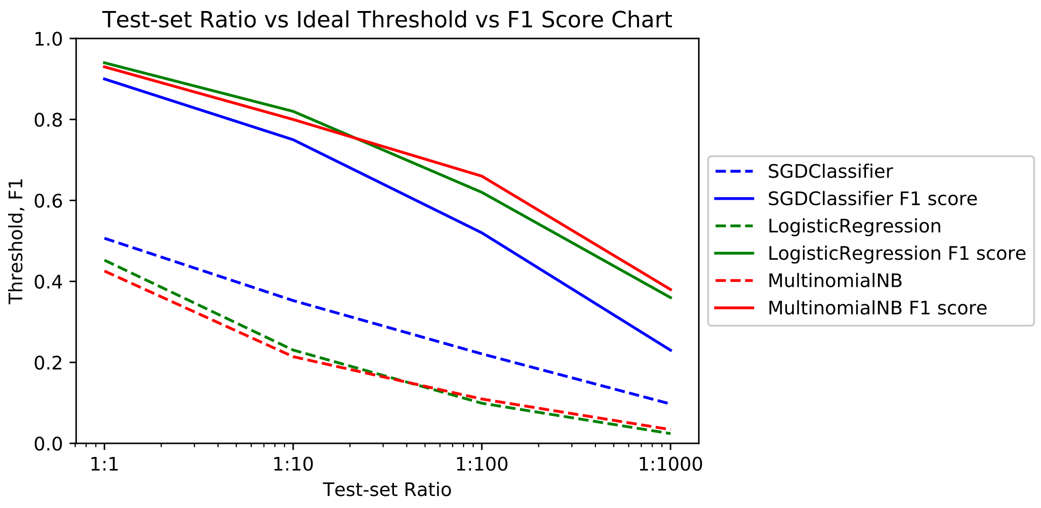


Figure 2.5: Ideal classification thresholds vs test set ratios vs F1-score for malicious class

Chapter 3

Model analysis

As mentioned in the introduction, many of todays ML systems suffer from inability to provide credible explanation of their conclusions.

Interpretability of the system is not about complete comprehention of the every single detail of the model for all of the data, it is about understanding classifiers judgement for the given case. In general, interpretability leads to more accurate verdicts and to a model that generalizes better [19]. The combination of sturdy and unbiased data base, explainable model and problem undestanding is necessary to create rubust ML solution, that performs on real-world data.

For instance, in healthcare, finance or automotive industries it is crucial to be able to audit/review the desision process and guarantee it is not prejudicial or violating any laws [19]. Furthermore, in safety-critical systems like medical applications or self-driving cars, a single wrong prediction can have a significant impact. Therefore it is critical to be able to verify the model and system should be able to explain how it reached a given recommendation. Such demand on interpretability will only growth with the rise of the enforcement of privacy and data protection regulation laws like CCPA or GDPR and quality solution becomes even more essential.

To solve this problem, one possibility is to use explainable models (see table 1.1) such as "decision tree" but large tree will not give us an answer what the overall logic of the system was or which feature was more important. Second option is to analyze data and construct model with meaningfull data properties in mind - disadvantages of this solution are mentioned in the section 1.2.5.

The other options are to analyze the data used for training and explain model through selection of meaningful input and output characteristics, to build build an explainable model using statistical methods and tthe last one is to analyze the dependencies between the input, output, and intermediate information about model. To the last group belongs model *interprettation frameworks* like **ELI5**, **Skater** or **SHAP** (Shapley additive explanations). The one to be presented is called **LIME**.

3.1 Local Interpretable Model-agnostic Explanations

LIME (Local Interpretable Model-agnostic Explanations) is a general framework that explains the predictions of any machine learning model in an interpretable manner. Basically is LIME trying to fit a linear model into a complex non-linear one. In order to achieve model indepence, it looks on the training dataset, on the outcome and gives result why model decided as such, modifying the input to the model locally, while treating ML model as a black-box [39]. This means, that explanatory model is not trying to undestand the entire model at the same time, instead an input instance is tweaked and the impact on the output is monitored [19]. Ergo, in the context of text classification, some of the words are replaced or modified, to determine which elements of the input impact the predictions [19].

In conclusion, explanation provided by LIME is local to the given query, interpretable for an end user to understand and model-agnostic, which means that explanation system is independent on the undelying ML model.

By explaining a prediction is meant to present textual or visual output that provide undestanding of connection between components of the output and the model's prediction [39]. To illustrate the situation, imagine that patient has a pertussis (also known as whooping cough or 100-day cough), but ML model predicts diagnosis that patient has a flu. The early symptoms of this disease are the same as the ones of the flu. LIME highlights the symptoms in the patient's history that led to the prediction. Runny nose and fever are interpretted as contributing to the "flu" prediction, while strong cough is indication, that estimated diagnosis might be wrong. With those data specialist can make an informed decision whether or not to trust the prediction and make additional tests to confirm diagnosis, which can prevent future complications.

In the paper named *"'Why Should I Trust You?' Explaining the Predictions of Any Classifier"* the explanation produced by LIME is expressed by the equation (3.1) [39].

$$\xi(x) = \underset{g \in G}{\arg\min}\, \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{3.1}$$

They define explanation as a model $g \in G$, where G is a set of interpretable models. Model g is in state to be demonstrated to the user using visual or textual elements. Because of the possibility, that $g \in G$ is not simple enough to be easily interpretable, measure of complexity $\Omega(g)$ is introduced.

> For example, for decision trees $\Omega(g)$ may be the depth of the tree, while for linear models, $\Omega(g)$ may be the number of non-zero weights [39].

Model being explained is denoted $f : \mathbb{R}^d \to \mathbb{R}$, while $f(x)$ is the transition function whose output can be used as a label for the explanation model. Expression $\pi_x(z)$ is a proximity measure between an instance z to x, to define locality around x. Let us express $\pi_x(z)$ as an exponential kernel (a window function) defined on some distance function D (e.g. cosine distance for text) with width σ, equation (3.2).

$$\pi_x(z) = \exp\left(\frac{-D(x,z)^2}{\sigma^2}\right) \tag{3.2}$$

If x and z are row vectors, their cosine similarity k is defined by equation (3.4). On L2-normalized data, this function is equivalent to *linear kernel* (3.3) for column vectors, only slower.

$$k(x,z) = x^\intercal z \tag{3.3}$$

This is called cosine similarity, because Euclidean normalization projects the vectors onto the unit sphere, and their dot product is then the cosine of the angle between the points denoted by the vectors [36].

$$D \triangleq k(x,z) = \frac{xz^\intercal}{\|x\| \cdot \|z\|} \tag{3.4}$$

In the following example usage of LIME framework, I am using TF-IDF vectorizer which will give us L2 normalized data.

Interpretable representation used for text classification can be described as a binary vector indicating the presence or absence of a word. Let $x \in \mathbb{R}^d$ be the original vector representation of a word and $x' \in \{0,1\}^{d'}$ to be binary vector of interpretable representation. Algorithm implemented by the framework samples instances around x' by taking nonzero elements of x'. In the paper they denote such vector as $z' \in \{0,1\}^{d'}$.

$$x' = \begin{bmatrix} 0 & 1 & 1 & 1 \end{bmatrix} \Rightarrow \begin{matrix} z'_1 = \begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix} \\ z'_2 = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \\ z'_2 = \begin{bmatrix} 0 & 1 & 0 & 1 \end{bmatrix} \\ \dots \end{matrix} \tag{3.5}$$

After this transformation, the original representation $z \in \mathbb{R}^d$ is obtained and $f(z)$ is acquired. The $f(z)$ is obtained using LASSO (least absolute shrinkage and selection operator) linear regression algorithm with x'_i as features and $f(z)$ as target.

When we let G be the class of linear models, such that $g(z') = w_g \cdot z'$, we can define $\mathcal{L}$ as locally weighted square loss function (3.6).

$$\mathcal{L}(f, g, \pi_x) = \sum_{z,z' \in \mathcal{Z}} \pi_x(z) \left(f(z) - g(z')\right)^2 \tag{3.6}$$

3.1.1 LIME evaluation

Solution provided by LIME presents locally faithful explanations. Those explanations are obtained by fitting linear model into more complex one. Linearization in such manner enables to achieve interpretability even though the original model might be to complex to explain globally. Locality used for explaining is captured by π_x, which in combination with *exponential kernel* leads to robustness to sampling noise.

Thanks to its flexibility this framework can be used for explaining text classification (e.g. Random Forests, Naive Bayes), image classification (e.g. neural networks) and regression.

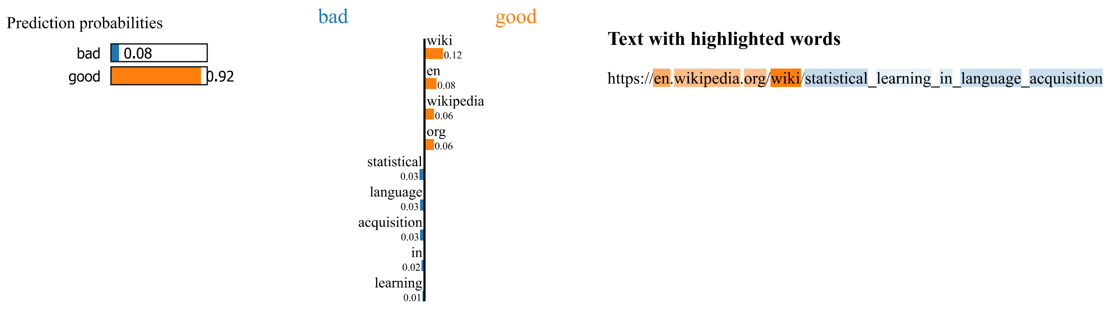


Figure 3.1: LIME explanation for given URL (text classification)

In order to undestand how LIME performs on the test data, an performed an experiment. For every URL in the testing set, the explanation procedure was executed. The output consists of list of URL features and number denoting whether feature supports or contradicts class affiliation. In our case, if the number is greater than zero, the feature supports benign class and vice versa. If the number is zero, it means that the feature was not present in the training data, therefore LIME is not able to make any assumptions based on that.

The goal of the experiment was to figure out the most important features. Each item in the testing data set containing 300 malicious and 3000 non-malicious URLs was evaluated, and the feature weights were counted. The process is not suitable for much larger sets, because the explanation retrieval can be time consuming - taking roughly one second to get the explanation for an URL - taking about one hour to process the entire set of 3300 URLs for one classifier. The counting was done in a four ways:

1. **cumulative** - both positive and negative numbers were added up,

2. **absolute** - the absolute value of numbers was added up,
3. **malign** - only negative numbers were added up,
4. **benign** - only positive numbers were added up.

	Cumulative		Absolute		Malign		Benign	
Classifier	Token	Value	Token	Value	Token	Value	Token	Value
SGD	html	86.42	html	86.42	net	−23.01	html	86.42
	htm	28.04	htm	28.04	php	−20.26	htm	28.04
	org	25.61	org	25.61	a	−19.70	org	25.61
	watch	21.26	net	23.01	index	−8.95	watch	21.26
	youtube	18.79	watch	21.26	2	−8.59	youtube	18.79
	people	18.30	php	20.26	1	−6.60	people	18.30
	wiki	16.99	a	19.70	i	−6.56	wiki	16.99
	the	16.21	youtube	18.79	co	−6.20	the	16.21
	2011	15.46	people	18.30	f	−6.08	2011	15.46
	ca	15.39	wiki	16.99	x	−5.95	ca	15.39
LRC	html	83.94	html	83.94	net	−16.16	html	83.94
	htm	30.45	htm	30.45	a	−10.98	htm	30.45
	watch	23.63	watch	23.63	index	−10.87	watch	23.63
	youtube	23.47	youtube	23.47	php	−9.53	youtube	23.47
	org	19.78	org	19.78	2	−6.54	org	19.78
	people	17.66	people	17.66	x	−5.98	people	17.66
	2011	16.52	2011	16.52	info	−5.96	2011	16.52
	wiki	13.16	net	16.16	co	−5.65	wiki	13.16
	ca	13.05	wiki	13.16	in	−5.10	ca	13.05
	life	13.02	ca	13.05	ru	−5.02	life	13.02
MNB	youtube	21.16	youtube	21.16	a	−6.76	youtube	21.16
	watch	19.61	watch	19.61	net	−6.01	watch	19.61
	html	13.15	html	13.15	php	−4.64	html	13.15
	wiki	10.54	wiki	10.54	d	−4.07	wiki	10.54
	people	9.90	people	9.90	f	−3.57	people	9.90
	2011	9.41	2011	9.41	b	−3.44	2011	9.41
	wikipedia	9.17	wikipedia	9.17	x	−3.40	wikipedia	9.17
	org	8.49	org	8.49	3	−3.39	org	8.49
	montreal	7.26	montreal	7.26	2	−3.20	montreal	7.26
	facebook	7.23	facebook	7.23	ru	−3.08	facebook	7.23

Table 3.1: The most valuable tokens for each classifier

In the table 3.1 is ten most valuable tokens according to each of the four metrics and all three classifiers. I may be not surprising that the **cumulative** and **benign** metrics show the same results. Those two may become interesting when analyzing dataset containing entities constructed e.g. for *evasion* attack.

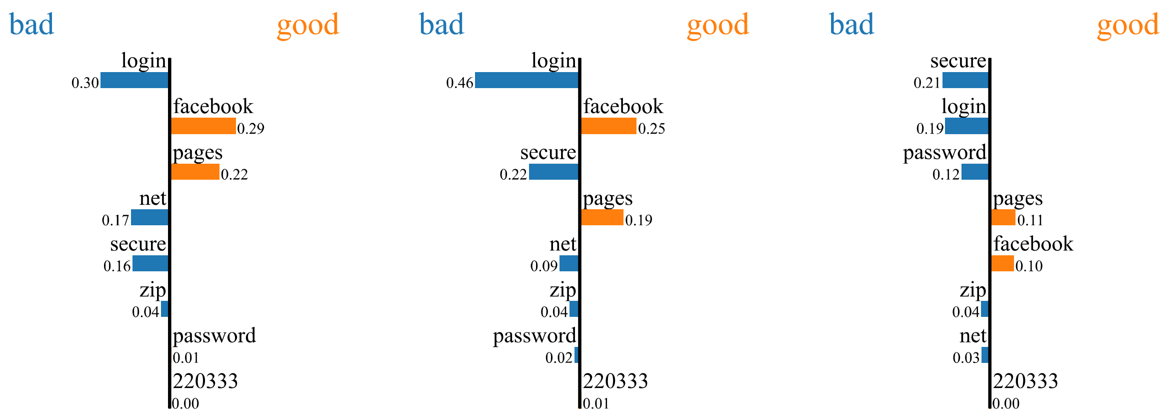
The **malign** metric reveals structure of malicious URLs in the dataset - single characters, file extensions and suspicious TLDs occupying the “top ten”. The **absolute** metrics is just a combination of those already mentioned.

More diverse and larger testing dataset may show more interesting results, although the process is very time consuming therefore unsuitable for frequent and comprehensive analysis.

The results for *Linear Regression* and *Stochastic Gradient Descend* are very similar, the *Multinomial Naive Bayes* has some differences (order, occurrence of words and overall lower values).

3.1.2 LIME and URL classification

For the testing purpose, I constructed purely fictional URL address https://www.facebook.net/pages/220333-zip-secure/passwordlogin. The URL is intended to be a malicious one, with strongly polarized expressions, which I observed empirically. For example, *facebook* always contribute to the benign decision, whereas *login* or *secure* make decision shift to the malign side.



(a) SGD (b) LRC (c) MNB

Figure 3.2: All explanations generated by LIME

Prediction probabilities that the URL belongs to the malicious class are in the table 3.2, the probability that the URL belongs to the class *good* is complementary to one.

Classifier	Probability
SGD	0.66
LRC	0.93
MNB	0.95

Table 3.2: Prediction probabilities for all classifiers that the example URL address is malicious

From the results, it seems that LIME explainer works correctly. The infomation we get from the explanations are not really usefull for real-life URL classification, but can be used for model tuning during development.

Suppose that we are trying to create a model, resistant to evasion attack. The way of feature extraction presented in this paper, does not allow to create a comprehensive *blacklist* of forbidden features. The URLs tend to be quite short, therefore every feature counts. LIME can help us to identify which features manifest themselves the most. Based on this information an engineer can adjust the model parameters in a way to lower the weights of the most significant attributes - making it harder for an adversary to create adversarial samples, hopefully improving model objectivity at the same time. The same information can be used to find possible weaknesses in data distribution, such as too many similar entities in the training data or missing features.

Model explanations might be very usefull (even necessary) in some situations and can be used to “sanity check” our trust in model decision and our belief in constructed classifiers.

Chapter 4

Conclusion and future work

4.1 Future work

In the paper I opened many topics, unable to explore them in depth. There are many possible ways, the paper could be augmented.

4.1.1 Improve URL tokenizer

The tokenizer now works only for english language. The model splitting a text string without spaces into tokens is trained on data from english Wikipedia and other language than english will not be tokenized properly. It would be handy to add possibility to tokenize several languages. I can see two possibilites:

1. update dataset with the language of the URL,
2. get language from TLD (Top level domain).

The first proposal would place high demands on the dataset management - conceivably solvable with ML model. The second proposal would be probably very low accurate.

There is another problem with current tokenizer - it does not split correctly brand names. To solve this problem, a whitelist of words which should be taken as they are should be made - for example *kaggle* (well known website with ML challenges and datasets), now tokenizer will not recognize as one word.

4.1.2 Multiclass classification

The idea here is to improve dataset labels in order to enable multiclass classification. The addresses could be labeled as malware (spyware, adware, ransomware, etc.), fakenews, gambling, etc., making multiclass explanations more interesting.

4.1.3 Model-nonagnostic explainer

Model agnosticism is a strong point of LIME framework. It was a necessity to develop framework like this in a model independent way. The idea behind creating a model-nonagnostic explainer is to “borrow” some data from the model and make explanations more accurate and more meaningfull.

4.1.4 Explanations using n-grams

The current state of LIME framework does not support n-gram and restrict the explanation terms to unigrams. This is drawback for many possible usages and augmenting the current implementation with this functionality would be very beneficial.

4.1.5 Compare explanation frameworks

The LIME is not the only framework, providing explanations of ML systems, available. There are others like ELI5, Skate or SHAP. I suggest an extension to this paper, which would serve as an status overview of those frameworks, comparison of their capabilities and use cases.

4.2 Conclusion

The use of machine learning in cybersecurity is a very complex topic becoming increasingly important. The first contribution of this paper is in the comprehensive overview of the current state of the machine learning from three points of view:

1. utilization of machine learning for systems defense used by security companies to mitigate security breaches,

2. summary of attacks which machine learning models are facing and possible ways of defense, including security of system infrastructure and training data,

3. ways how could an adversary take an advantage of machine learning to cause damage to an infected system.

I reached a conclusion that even though those methods are integral part of security systems, they cannot be relied upon completely, making human surveillance and traditional ways of threat recognition still irreplaceable. Furthemore, despite of the growing numbers of "prove of concept" papers on the topic of use of machine learning for attacks, it seems that those methods are not much widespread, if at all.

There are plenty of ways how to collect URL features, making this problem very context dependent. The next contribution is in the presented way of feature extraction from URL addresses - proving that inferring spaces from URL is possible and although the number of inferred words is very limited and in some cases very inaccurate, the classification is still attainable. The weakness of this section is at the data. In real application scenario, it will be impossible to train a classifier on the datasets freely available on the internet - it is necessary to observe and extract data from real-time network traffic. On the other hand the used classiers are very appropriate for the task, working quite well even without additional hyperparameter tuning.

The last chapter is devoted to the model analysis and it is the last contribution of this work - making sense of obtained URL classifications. Explaining the predictions of any machine learning classifier will become more important after machine learning systems become widely used in healthcare and other decision sensitive applications. It is quite complicated to compare *explainer* output from several classifiers, insomuch as final explanation is highly dependent on used machine learning algorithm. Overall, the results are understandable and help to accept and build trust in machine learning models.

Appendix A

Source code

Source code used in the thesis is available at https://gitlab.com/mareklovci/bachelors-thesis.

Appendix B

URL tokenizer

Implementation of algorithm infering the location of spaces in a string as presented in the original *StackOverflow* question [16]. The code is written at the Python programming language.

```python
from math import log

words = open("words-by-frequency.txt").read().split()
wordcost = dict((k, log((i+1)*log(len(words)))) for i,k in enumerate(words))
maxword = max(len(x) for x in words)


def infer_spaces(s):

    def best_match(i):
        candidates = enumerate(reversed(cost[max(0, i-maxword):i]))
        return min((c + wordcost.get(s[i-k-1:i], 9e999), k+1) for k,c in candidates)

    # Build the cost array.
    cost = [0]
    for i in range(1,len(s)+1):
        c,k = best_match(i)
        cost.append(c)

    # Backtrack to recover the minimal-cost string.
    out = []
    i = len(s)
    while i>0:
        c,k = best_match(i)
        assert c == cost[i]
        out.append(s[i-k:i])
        i -= k

    return " ".join(reversed(out))
```

Acronyms

API	Application Programming Interface. 7
CAPTCHA	Completely Automated Public Turing tests to tell Computers and Humans Apart. 10
CVE	Common Vulnerabilities and Exposures. 5
DLL	Dynamic-link library. 2
IPS	Internet Provider Security tag. 2
PDF	Portable Document Format. 2
URL	Uniform Resource Locator. 13

Bibliography

[1] *Artificial Intelligence for Cybersecurity: The American Institute of Aeronautics and Astronautics*. Apr. 18, 2019. URL: https://www.aiaa.org/protocolAI/ (visited on 04/18/2019).

[2] Avira Operations GmbH & Co. KG. *NightVision – Using Machine Learning to Defeat Malware*. Tech. rep. 2018. URL: https://oem.avira.com/resources/whitepaper_NightVision_EN_20180306.pdf.

[3] Battista Biggio et al. "Evasion Attacks against Machine Learning at Test Time". In: *CoRR* abs/1708.06131 (2017). arXiv: 1708.06131. URL: http://arxiv.org/abs/1708.06131.

[4] Jason Brownlee. *Logistic Regression for Machine Learning*. Apr. 2016. URL: https://machinelearningmastery.com/logistic-regression-for-machine-learning/ (visited on 05/13/2019).

[5] Ebubekir Büber. *Phishing URL Detection with ML*. Feb. 2018. URL: https://towardsdatascience.com/phishing-domain-detection-with-ml-5be9c99293e5 (visited on 02/25/2019).

[6] Clarence Chio. *Machine learning and security: protecting systems with data and algorithms*. Sebastopol, CA: O'Reilly Media, 2018. ISBN: 978-1491979907.

[7] CrowdStrike, Inc. *The Rise of Machine Learning in Cybersecurity*. Tech. rep. URL: https://go.crowdstrike.com/rs/281-OBQ-266/images/WhitepaperMachineLearning.pdf.

[8] Elliot J Crowley et al. "Pruning neural networks: is it time to nip it in the bud?" In: *arXiv preprint arXiv:1810.04622* (2018).

[9] *Cyber Attack Lifecycle - Law Enforcement Cyber Center*. Apr. 27, 2019. URL: http://www.iacpcybercenter.org/resource-center/what-is-cyber-crime/cyber-attack-lifecycle/ (visited on 04/27/2019).

[10] Darktrace Limited. *The Next Paradigm Shift: AI-Driven Cyber-Attacks*. Tech. rep. 2018. URL: https://www.darktrace.com/en/resources/wp-ai-driven-cyber-attacks.pdf.

[11] *Differential Privacy and IP Protection Solutions from CryptoNumerics*. Apr. 30, 2019. URL: https://cryptonumerics.com/privacy-protecting-products/cn-protect/ (visited on 04/30/2019).

[12] *DNS Security with DNSCrypt / OpenDNS*. Mar. 14, 2019. URL: https://www.opendns.com/about/innovations/dnscrypt/ (visited on 05/13/2019).

[13] Sumit Dua. *Text Classification using K Nearest Neighbors – Towards Data Science*. Nov. 2018. URL: https://towardsdatascience.com/text-classification-using-k-nearest-neighbors-46fa8a77acc5 (visited on 05/13/2019).

[14] ESET, spol. s r.o. *Is machine learning cybersecurity's silver bullet?* Tech. rep. URL: https://i.crn.com/sites/default/files/ckfinderimages/userfiles/images/crn/custom/2018/ESET_360_Q318_NextGen_Whitepaper.pdf.

[15] Ian Goodfellow and Nicolas Papernot. *The challenge of verification and testing of machine learning | cleverhans-blog.* June 2017. URL: http://www.cleverhans.io/security/privacy/ml/2017/06/14/verification.html (visited on 04/08/2019).

[16] *How to split text without spaces into list of words? - Stack Overflow.* Apr. 29, 2019. URL: https://stackoverflow.com/questions/8870261/how-to-split-text-without-spaces-into-list-of-words/11642687#11642687 (visited on 04/29/2019).

[17] Weiwei Hu and Ying Tan. "Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN ". In: *CoRR* abs/1702.05983 (2017). arXiv: 1702.05983. URL: http://arxiv.org/abs/1702.05983.

[18] Huawei Technologies Co., Ltd. *AI Security White Paper.* Tech. rep. 2018. URL: https://www-file.huawei.com/-/media/corporate/pdf/cyber-security/ai-security-white-paper-en.pdf?la=en-gb.

[19] Lars Hulstaert. *Interpreting machine learning models.* Feb. 2018. URL: https://towardsdatascience.com/interpretability-in-machine-learning-70c30694a05f (visited on 03/17/2019).

[20] Roberto Iriondo. *Breaking CAPTCHA Using Machine Learning in 0.05 Seconds.* Oct. 19, 2018. URL: https://medium.com/mlmemoirs/breaking-captcha-using-machine-learning-in-0-05-seconds-9feefb997694 (visited on 04/23/2019).

[21] Luke Irwin. *How long does it take to detect a cyber attack? - IT Governance USA Blog.* Mar. 14, 2019. URL: https://www.itgovernanceusa.com/blog/how-long-does-it-take-to-detect-a-cyber-attack (visited on 04/18/2019).

[22] Jay Jacobs. *Building a DGA classifier.* Sept. 2014. URL: https://datadrivensecurity.info/blog/posts/2014/Sep/dga-part1/ (visited on 02/25/2019).

[23] Michal Kosinski, David Stillwell, and Thore Graepel. "Private traits and attributes are predictable from digital records of human behavior". In: *Proceedings of the National Academy of Sciences* 110.15 (2013), pp. 5802–5805. ISSN: 0027-8424. DOI: 10.1073/pnas.1218772110. eprint: https://www.pnas.org/content/110/15/5802.full.pdf. URL: https://www.pnas.org/content/110/15/5802.

[24] Ryan Kovar. *Random Words on Entropy and DNS.* Oct. 2015. URL: https://www.splunk.com/blog/2015/10/01/random-words-on-entropy-and-dns.html (visited on 02/25/2019).

[25] Kaspersky Lab. *Going beyond Next Gen Security - Is your business getting the protection it deserves?* Tech. rep. 2018. URL: https://media.kaspersky.com/en/business-security/KES_Going_beyond_Next_Gen_whitepaper.pdf.

[26] Kaspersky Lab. *Machine Learning for Malware Detection.* Tech. rep. 2019. URL: https://media.kaspersky.com/en/enterprise-security/Kaspersky-Lab-Whitepaper-Machine-Learning.pdf.

[27] Ricky Laishram and Vir Virander Phoha. "Curie: A method for protecting SVM Classifier from Poisoning Attack". In: *CoRR* abs/1606.01584 (2016). arXiv: 1606.01584. URL: http://arxiv.org/abs/1606.01584.

[28] Jin-Lee Lee et al. "Heuristic-based Approach for Phishing Site Detection Using URL Features". In: 2015.

[29] Changming Liu. *What are DGAs?* Feb. 2018. URL: https://aelladata.com/2018/02/11/what-are-dgas/ (visited on 02/25/2019).

[30] Kang Liu, Brendan Dolan - Gavitt, and Siddharth Garg. "Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks". In: *CoRR* abs/1805.12185 (2018). arXiv: 1805.12185. URL: http://arxiv.org/abs/1805.12185.

[31] Yuntao Liu, Yang Xie, and Ankur Srivastava. “Neural Trojans”. In: *CoRR* abs/1710.00942 (2017). arXiv: 1710.00942. URL: http://arxiv.org/abs/1710.00942.

[32] *machine learning - explain meaning and purpose of L2 normalization - Cross Validated.* Mar. 6, 2018. URL: https://stats.stackexchange.com/questions/331926/explain-meaning-and-purpose-of-l2-normalization (visited on 03/21/2019).

[33] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Sch¨ u tze. *Introduction to Information Retrieval.* New York, NY, USA: Cambridge University Press, 2008. ISBN: 0521865719, 9780521865715.

[34] Christoph Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable.* 2019. URL: https://christophm.github.io/interpretable-ml-book/.

[35] Nicolas Papernot, Patrick D. McDaniel, and Ian J. Goodfellow. “Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples”. In: *CoRR* abs/1605.07277 (2016). arXiv: 1605.07277. URL: http://arxiv.org/abs/1605.07277.

[36] F. Pedregosa et al. “Scikit-learn: Machine Learning in P ython”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[37] Florian Tram è r et al. “Stealing Machine Learning Models via Prediction APIs”. In: *25th USENIX Security Symposium (USENIX Security 16).* Austin, TX: USENIX Association, 2016, pp. 601–618. ISBN: 978-1-931971-32-4. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/tramer.

[38] Roy Rahul. *ML | Stochastic Gradient Descent (SGD) - GeeksforGeeks.* May 13, 2019. URL: https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/ (visited on 05/13/2019).

[39] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “"Why Should I Trust You?": Explaining the Predictions of Any Classifier”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016.* 2016, pp. 1135–1144.

[40] Andrea Scarfo. *Domain generation algorithms – Why so effective?* Oct. 2016. URL: https://umbrella.cisco.com/blog/2016/10/10/domain-generation-algorithms-effective/ (visited on 11/29/2018).

[41] The Agency for Digital Italy. *Artificial Intelligence at the service of citizens.* Tech. rep. Mar. 2018. URL: https://ia.italia.it/assets/whitepaper.pdf.

[42] *Trickbot becomes one of the most dangerous pieces of modular malware hitting enterprises - Help Net Security.* Apr. 30, 2019. URL: https://www.helpnetsecurity.com/2019/02/14/trickbot-business-threat/ (visited on 04/30/2019).

[43] Yizhen Wang and Kamalika Chaudhuri. “Data Poisoning Attacks against Online Learning”. In: *CoRR* abs/1808.08994 (2018). arXiv: 1808.08994. URL: http://arxiv.org/abs/1808.08994.

[44] *What are popular text classification algorithms in commercial use and how are they used? - Quora.* May 13, 2019. URL: https://www.quora.com/What-are-popular-text-classification-algorithms-in-commercial-use-and-how-are-they-used (visited on 05/13/2019).

[45] *What is the difference between the the Gaussian, Bernoulli, Multinomial and the regular Naive Bayes algorithms? - Quora.* May 13, 2019. URL: https://www.quora.com/What-is-the-difference-between-the-the-Gaussian-Bernoulli-Multinomial-and-the-regular-Naive-Bayes-algorithms (visited on 05/13/2019).

[46] Rey Wiyatno and Anqi Xu. “Maximal Jacobian-based Saliency Map Attack”. In: *CoRR* abs/1808.07945 (2018). arXiv: 1808.07945. URL: http://arxiv.org/abs/1808.07945.

[47] Tianqing Zhu. *Explainer: what is differential privacy and how can it protect your data?* Apr. 30, 2019. URL: https://theconversation.com/explainer-what-is-differential-privacy-and-how-can-it-protect-your-data-90686 (visited on 04/30/2019).